MCCORMICK, PAULDING & HUBER LLP
City Place II
185 Asylum Street
Hartford, CT 06103-4102
Tel. (860) 549-5290

"EXPRESS MAIL" MAILING LABEL
NUMBER EV337023820US
DATE OF DEPOSIT December 4, 2003
I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 C.F.R. 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, MAIL STOP PATENT APPLICATION, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450

LISA M. TINNIRELLA
(TYPED NAME OF PERSON MAILING PAPER OR FEE)

Lisa M. Tinnirella
(SIGNATURE OF PERSON MAILING PAPER OR FEE)

Mail Stop **PATENT APPLICATION**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

ATTACHMENT TO A PATENT APPLICATION

DOCKET NO.:	5227-01-2
ENTITLED:	CELLULAR ENGINE FOR A DATA PROCESSING SYSTEM
INVENTOR(S):	GHEORGHE STEFAN and DAN TOMESCU
INCLUDING:	Specification; Claims; Abstract; and five (5) sheets of Informal Drawings

CELLULAR ENGINE FOR A DATA PROCESSING SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/431,154 entitled "ENHANCED VERSION OF CONNEX MEMORY", and filed on December 5, 2002, hereby incorporated by reference in its entirety. The subject matter of this application relates to pending U.S. Patent Application Serial No. 09/928,151 entitled "A MEMORY ENGINE FOR THE INSPECTION AND MANIPULATION OF DATA", filed on August 10, 2001 and U.S. Patent Application Serial No. _______________entitled "DATA PROCESSING SYSTEM FOR A CARTESIAN CONTROLLER", filed December ______, 2003, both of which are herein incorporated by reference in their entirety.

FIELD OF THE INVENTION

[0002] The invention relates generally to an engine for a data processing system, and more particularly, to a cellular engine for a data processing system that implements an active associative memory device, or associative engine, to increases data processing speeds and efficiency.

BACKGROUND OF THE INVENTION

[0003] Automated or semi-automated data processing systems are integral components in a wide variety of applications. Typically, data management systems are embedded within a larger computerized apparatus or system and serve to assist or facilitate those applications running in the larger computerized system, such as by performing necessary arithmetic operands, data conversion or the like.

[0004] As is known, basic data processing systems may be categorized as single instruction, single data stream (SISD) devices and typically utilize, in their simplest expression, a processor, an interface and a memory device. The processor performs directed tasks in response to instructions inputted either by a user, or by another component of an overall system. In performing its designated tasks, the processor relies upon the interface to communicate commands, such as data requests, to the memory device, as well as to receive thereby specified data stored within the memory device.

[0005] Known data processing systems most often utilize conventionally addressed memory devices. That is, known data systems utilize memory devices which include defined locales therein, each locale having its own particularized address. In this manner, should the processor desire to add the value stored at address A with the value stored at address B, the memory device will proceed to the specific, addressed locations, or cells, within the memory device, and communicate these values, via the interface, to the processor where the appropriate summation can occur. In such systems, the nature and capability of the integral components, that is, the nature and capabilities of the processor and the memory devices, are well defined and distinct from one another. Figure 1 depicts such a known data processing system wherein processor **2** operates in response to tasks inputted via input line **4**. An interface **6** is thereafter utilized to communicate instructions, such as data requests, to the memory device **8**, as well as to receive thereby specified data stored within the memory device **8**.

[0006] It is also known that data processing systems may include more than one processor and memory device, and further, that these multiple components may be part of a system that executes multiple streams of instructions. These multiple instruction streams, multiple data streams (MIMD) devices can be viewed as large collections of tightly coupled SISD devices where each processor in the system, although operating in overall concert with the other integrated processors, is responsible for a specific portion of a greater task. That is, the effectiveness of MIMD devices is typically limited to those specified arenas where the problem to be solved lends itself to being parsable into a plurality of similar and relatively independent sub-problems. The nature and capabilities of those integral components of MIMD devices are also well defined and distinct from one another.

[0007] Another known data processing system involves single instruction, multiple data streams (SIMD) devices. These SIMD devices utilize an arbitrary number of processors which all execute, in sync with one another, the same program, but with each processor applying the operator specified by the current instruction to different operands and thereby producing its own result. The processors in a SIMD device access integrated memory devices to get operands and to store results. Once again, the nature and capabilities of those integral components of a SIMD device are well defined and distinct from one another in that computations are executed by the processors that must have some type of access to a memory device to do their job.

[0008] While known data processing systems are therefore capable of processing large amounts of data, the defined and unchanging nature of the processors and memory devices limits the speed and efficiency at which various operations may be completed.

[0009] Various architectures have also been constructed which utilize another class of memory devices which are not conventionally addressed. These memory devices are typically described as being 'associative' memory devices and, as indicated, do not catalog their respective bits of data by their location within the memory device. Rather, associative memory devices 'address' their data bits by the nature, or intrinsic quality, of the information stored therein. That is, data within associative memory devices are not identified by the name of their locations, but from the properties of the data stored in each particular cell of the memory device.

[0010] A key field of fixed size, is attached to all data stored in most associative memory devices. A search key may then be utilized to select a specific data field, or plurality of data fields whose attached key fileld(s) match the search key, from within the associative memory device, irrespective of their named location, for subsequent processing in accordance with directed instructions.

[0011] While the implementation of associative memory devices is therefore known, these devices have always been utilized as specialized blocks, or components, within known data processing systems employing standard processors, interfaces and conventionally addressed memory devices. That is, although known associative memory devices do not employ conventional addressing protocols, they are incapable of processing the information themselves, relying instead upon known processors and external memory devices in a manner consistent with known SISD, SIMD and MIMD architectures.

[0012] With the forgoing problems and concerns in mind, the present invention therefore seeks to provide an engine for a data processing system that overcomes the above-described drawbacks by utilizing an active associative memory device using variable-size keys whose cells, by selectively acting as both a processor and a memory device, never have to access a separate memory block to do their jobs, thus substantially reducing processing, computational and communication times.

SUMMARY OF THE INVENTION

[0013] It is an object of the present invention to provide an efficient data processing system.

[0014] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active associative memory, or associative engine device, in a manner which increases data processing speeds and efficiency.

[0015] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active associative memory, or associative engine device whose cells, by selectively acting as both a processor and a memory device, never have to access a separate memory block to do their jobs.

[0016] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active associative memory device, or associative engine, whose individual cells can selectively process a given instruction based upon their respective state as set by a globally propagated instruction or query.

[0017] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active associative memory device, or associative engine, whose individual cells can selectively process, in parallel, a given instruction based upon their respective state, all within a single clock cycle.

[0018] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active memory device, or cellular engine, that allows the use of variable-length key fields.

[0019] It is another important aspect of the present invention to provide a cellular engine for a data processing system that implements an active memory device, or cellular engine, whose structure is homogeneous, thus allowing the very same piece of information stored in memory to be (part of) either the key field or data field at different times during the execution of a program.

[0020] It is another object of the present invention to provide a cellular engine for an efficient data processing system that enables the dynamic limitation of the search space within an active associative memory device.

[0021] It is another object of the present invention to provide a cellular engine for an efficient data processing system that provides for the selective accessibility of either end of the cell array.

[0022] It is another object of the present invention to provide an engine for an efficient data processing system which is capable of regulating data transmission between two or more cells within an associative memory device.

[0023] According to one embodiment of the present invention, a data processing system includes an associative memory device containing *n*-cells, each of the *n*-cells includes a processing circuit. A controller is utilized for issuing one of a plurality of instructions to the associative memory device, while a clock device is utilized for outputting a synchronizing clock signal comprised of a predetermined number of clock cycles per second. The clock device outputs the synchronizing clock signal to the associative memory device and the controller globally communicates one of the plurality of instructions to all of the *n*-cells simultaneously, within one of the clock cycles.

[0024] These and other objectives of the present invention, and their preferred embodiments, shall become clear by consideration of the specification, claims and drawings taken as a whole.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] Figure 1 is a block diagram illustrating a known SISD data processing architecture.

[0026] Figure 2 is a block diagram showing the general configuration of a data processing system, including a memory engine and a synchronizing clock element, according to one embodiment of the present invention.

[0027] Figure 3 is a block diagram showing a more detailed view of the memory engine shown in Figure 2.

[0028] Figure 4 is a block diagram showing the structure of a cell, or processing element, according to one embodiment of the present invention.

[0029] Figure 5 is a block diagram showing the structure of the transcoder.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0030] Figure 2 depicts the architectural relationship between a controller **100**, hereinafter referred to as the Cartesian Controller (CC), and a cellular engine **102**, hereinafter referred to as the Connex Engine (CE). A synchronizing clock circuit **106** is utilized to coordinate the operation of the CC **100** and the CE **102** such that one of a plurality of instructions may be issued by the CC **100** and transferred to the CE **102** for parallel execution and processing.

[0031] The clock circuit **106** is capable of outputting a predetermined number of clock cycles per second, and the CC **100** is capable of performing an internal operation such that the CC **100** may perform one of a plurality of internal operations while also issuing in parallel one of a plurality of instructions to the CE **102** within a single clock cycle.

[0032] As depicted in Figure 3, the CE **102** is made up of an array of active cells, or processing elements, embodied as the Connex Memory (CM) **104** and a RAM (random access memory) containing a plurality of vectors **108**, each vector having the same storage capacity as the CM **104** and thus being capable of selectively storing the entire contents of the CM **104**. That is, the CE **102** includes the CM **104** having *n*- cells and the associated vector memory **108**, which is under the control of the sequential CC **100**. In one embodiment of the present invention, the purpose of the memory vectors **108**, is to allow for search, insert and delete operations to be performed on character strings longer than may be accommodated within the CM **104**, and to offer a lower cost of implementation and reduced power dissipation, as will be discussed in more detail later.

[0033] It will be readily appreciated that the present invention contemplates that the CE **102** may have any number of circuit-specific configurations without departing from the broader aspects of the present invention provided that the CC **100** is capable of issuing commands to, and receiving data from, the CE **102**.

[0034] Referencing Figure 3, each n-bit cell in the CM **104** contains the following registers:

- 'mark' - a one-bit marker register;
- 'w' - the main register, which plays a role similar to that of an accumulator in a more conventional design, while also being associated with a number of general registers equal to the number of memory vectors.

[0035] **Notations:** Let x be a bit field of length m and y be a bit field of length n: {x, y} denotes a bit field of length $m + n$ formed by appending y to x. The notation can be generalized to any number of bit fields, e.g. for three arguments: {a, b, c} = {a, {b, c}} = {{a, b}, c}.

[0036] Let r be an (n + 1)-bit register and $n \geq k \geq 0$: r[k] denotes the (k + 1) - *th* bit in r, counting from the right end (k = 0) to the left end (k = n).

[0037] Let r be an (n + 1) bit register and $n \geq m > k \geq 0$: r[m:k] denotes a bit field of length m-k+1, {r[m], r[m-1], . . . , r[k]}.

[0038] Therefore, the contents of an m- bit cell within the CM **104** are the contents of its w register appended to the contents of its *mark* register:

cell [m-1:0] = {mark, value}

value [m-2:0] = {ext, symbol}

symbol [m-3:0]

where *ext* stands for the extension bit, used to create an alphabet of special values (for *ext* = 1).

[0039] An important aspect of the present invention resides in the ability of each m-bit cell within the CM **104** to actively process data in addition to storing data. The processing of data may occur either within each m-bit cell, or by affecting the cell immediately to the left or right of a predetermined cell. It should be noted that by enhancing the functionality of each m-bit cell within the CM **104** in this manner, the present invention exhibits a system-level behavior that is more complex and, as such, exceeds the performance of other data processing systems.

[0040] It is another important aspect of the present invention that the ability to actively process data at the cell level within the CM **104** is accomplished, in part, by the ability of each cell to be 'marked', which is part of the condition, or predicate, designating it as a cell which will subsequently perform a task, or execute an operand, on itself, or an adjacent cell within the CM **104**.

[0041] Returning to Figure 3, for the purposes of the present invention, a cell is considered **'marked'** if the one-bit, internal cell register *mark* = 1 and is considered **'not marked'** if *mark* = 0. Moreover, the CM **104** has a **'left limit'** and a **'right limit'** that can be dynamically adjusted by issuing specialized instructions. In addition, the **'search space'** of the CM **104** is that segment of the CM **104** which is delimited by the left limit and the right limit. It will be readily appreciated that the internal cell register *mark* may be greater than one bit in length without departing from the broader aspects of the present invention.

[0042] As mentioned previously, the execution of code supplied to the CE **102** is driven by the CC **100**. The CE **102**/CC **100** interface makes use of four special registers, as shown in Figure 3:

- 'INR' **112** - data input register - all CE **102** instructions get their (immediate) data argument (if any) from INR (supplied by CC 100);
- 'OUTR' **114** - data output - contains the *'no mark'* bit and a value. If at least one of the cells is marked, OUTR contains 0 followed by the value contained in the first marked cell; otherwise OUTR contains 1 followed by an implementation-dependent special value, such as 11...1;
- 'OPR' **116** - instruction register, contains the operation code for the current CE 102 instruction (the source is a dedicated field in the CC **100** instruction);
- 'VAR' **118** - address register for the vector memory. The VAR **118** register is updated by special CC **100** instructions and is used as an argument to instructions that explicitly manipulate vectors; The VAR **118** register is also used

in the execution of all operations involving the general registers associated with cells.

[0043] As further represented in Figure 3, input/output lines **120** may be selectively utilized to access both ends of the CM **104**. As utilized herein, the input/output lines **120** have the following meaning:

- 'left_in' **122** = {w, mark, eq, first}, by default all are 0 (eq = 1 means that the two operands in the cell are equal; first = 1 means the cell is the first marked cell);
- 'left_out' **124** = {w, mark, eq, first}, come from the first cell;
- 'right_in' **126** = {w, mark, eq, first}, by default all are 0;
- 'right_out' **128** = {w, mark, eq, first}, come from the last cell.

[0044] Figure 4 illustrates one embodiment of an internal structure of the *m*-bit cells within the CM **104**. As shown in Figure 4, the internal structure of each cell includes the following circuits:

- **ALU**: arithmetic and logic unit **130** that performs addition, subtraction, comparisons, and bitwise logic functions
- **leftMux**: multiplexer **131** which selects the left operand for **ALU** from:
  - **w**: the value stored in the accumulator register **134**
  - **in**: the value received from the input register **112**
  - **memOut**: the value read from the vector memory addressed by the vector address register, **VAR 118**
- **aluMux**: multiplexer **133,**which selects the value to be loaded into the accumulator register (**w 134**) from:
  - **w**: the value stored in the accumulator register **134**
  - **fromLeft**: the value stored in the accumulator register of the left cell
  - **fromRight**: the value stored in the accumulator of the right cell

- the output of **ALU 130**
- **w**: the accumulator register **134**
- **mark**: the marker register
- tristate output buffers
- **DECODE**: a combinational circuit **137** which decodes the instructions according to the local context generated by:
  - **localFlags**: generated by **ALU**
  - **leftFlags**: the flags received from the left cell
  - **rightFlags**: the flags received from the right cell
  - **class_i**: classification code received from the **TRANSCODER** generating:
    - command codes for **leftMux, rightMux, ALU, aluMux, mark, tristate buffers**
    - flags from neighboring cells
    - **state_i** bit of the cell for the **TRANSCODER**

[0045] **The Transcoder**

[0046] Figure 5 illustrates the organization of the **TRANSCODER**, a circuit integrated with the CE **102** and acting as part of a control interconnection network. That is, the **TRANSCODER** is utilized, in part, to classify each cell according to:

- its state bit (**state_i**), (i.e. the local state)
- the state bits of all cells, (i.e. the global state)
- the current instruction to be performed,

into the following categories:

- marked cell
- first marked cell
- last marked cell
- cell within limits (the limits are stored in a memory area of the **TRANSCODER**)
- active cell.

[0047] It will be readily appreciated that other cell categories could be added to the above without departing from the broader aspects of the present invention.

[0048] The **TRANSCODER** receives from each cell a state bit (**state_0, state_1, ..., state_(n-1)**) and sends back to each cell a 3-bit code specifying the class(es) to which it belongs (**class_0, class_1, ..., class_(n-1)**).

[0049] The building blocks of the transcoder are:

- **OR Prefixes**: a circuit **140** calculating mainly the relative positions of cells according to the classification to be performed
- **Limit memory**: two latches **141** used to store information about the limits
- **MUXs**: multiplexers **142** which, based on the current instruction, select the class of each cell

[0050] The **TRANSCODER** can be implemented in two different ways in order to optimize its size and speed:

- a linear version for small values of *n*
- a bi-dimensional version (for large values of *n*), containing a **LINE TRANSCODER** and a **COLUMN TRANSCODER**, each having a size of the order $O(n^{1/2})$.

[0051] Another important aspect of the present invention, therefore, is the ability of the CC **100** to issue instructions to the CE **102** and cause such instructions to be broadcast, in parallel and in a single clock cycle, to all cells within the CM **104**. Those cells meeting the criteria set out in the instructions from the CC **100** may, for example, independently and selectively mark themselves, simultaneously in the same clock cycle, whereby subsequent instructions or operations, in the following clock cycle, may be effectuated according to the resulting classification, again in parallel and in a single clock cycle.

[0052] It is therefore another important aspect of the present invention that the **TRANSCODER** not only classifies each cell in accordance with its local state, for example, its marked or non-marked state, but also in accordance to its global state and the current instruction. That is, while one aspect of a cell's classification by the TRANSCODER may be that a particular cell's local state is 'marked', it is also important for such a cell to 'know' its 'global state' with respect to all other cells, such as whether the 'marked' cell is the 'first marked' cell or the 'last marked' cell.

[0053] By way of an example, suppose certain cells within the CM **104** have been marked, via some property of these cells as indicated by an instruction from the CC **100** in the previous clock cycle, as follows: (marked cells being represented by bolded numbers in the string):

CM: **2** **5** 2 7 6 **4** 10 ...

[0054] Suppose next that the instruction "addr 5" is broadcast to all cells within the CM **104**, again in parallel and in a single clock cycle, where vector 5 in the vector memory **108** is as follows:

Line 5: 3 4 7 8 2 5 12 ...

[0055] All marked cells within the CM **104** will then add the contents of their data field to the contents of the corresponding element in vector 5, with the result of this operation being stored in the respective cells of the CM **104**, as follows:

CM: **5** **9** 2 7 6 **9** 10 ...

[0056] As indicated by the example above, the marked/non-marked state of each cell within the CM **104** is not affected by the particular instruction issued by the CC **100** (in this example; although as noted, certain instructions will affect the marked state of each cell within the CM **104**). Moreover, all addition operations are executed simultaneously (that is, in parallel with one another) and internal to each marked cell within a single clock cycle.

[0057] As further indicated by the example above, the data processing system of the present invention can implement, at the system level, operations defined on vectors of values; in this example the vector values are the CM **104** data and vector 5 of the vector memory **108**. In this regard, the data processing system of the present invention includes a CE having a CM **104** with a linear array of *active* cells (i.e., processing elements) where each cell within the CM **104** has one or more marker bits and one accumulator **(134)**; at same time, at the level of each cell, the corresponding elements of all vectors can be seen as a set of associated registers. (the number of associated registers is therefore equal to the number of vectors **108**).

[0058] Moreover, it is another important aspect of the present invention that by concatenating the accumulators and individual associated registers of each cell within the CM **104** respectively, the data processing system of the present invention provides for operations on vectors of values, thereby enabling matrix computations and the like.

[0059] As contrasted to SIMD and MIMD systems, discussed in the Background of the present invention, the data processing system of the present invention does not rely upon an unchanging and strict delineation between the operations of a processor and a linked memory device.

[0060] In the data processing system of the present invention, information may be stored for retrieval in the CM **104**, but conventional "memory address' is not a concept; although each cell within the CM **104** may itself selectively process information in response to globally issued commands, it is not a processor *per se*, although each cell within the CM **104** does contain a processing circuit, as discussed previously; and the performance of the data processing system of the present invention is strictly a linear function of its size and applicable across a wide range of programs/applications in a manner that is not exhibited by other known programmable machines.

[0061] It will be readily appreciated that the data processing system of the present invention is not limited in the nature and configuration of the processing circuit contained within each of the *n*- cells in the CM **104**.

[0062] In accordance with the present invention, the CE **102** has a rich set of instructions, grouped in the following classes:

- **global management** - setting and resetting the CM **104** limits; exchanging data between the CM **104** and RAM vectors **108**;
- **search/access** - associative access to one or more of the cells of the CM 104;
- **marker manipulation**;
- **data store and transfer**;
- **arithmetic and logic**;
- **conditional**; and
- **index**.

[0063] Note that all instructions for the CE **102** are executed in a single machine cycle.

[0064] The CE **102** does not itself have access to a program memory to fetch its instructions - every cycle its cells expect to get an operation code in a special register, but it takes a different entity, in the present case, the CC **100**, to do it for them; the code is sequential and there needs to be a single point of access to fetch it. The main job of the CC **100** is therefore to drive the execution of the programs of the CE **102**, i.e., fetch instructions to be executed by individual cells and place them in an internal register; at the same time, it serves as a gateway to CE **102** and thereby takes care of all input/output interactions. The CC **100** also executes simple sequential operations without which it would be impossible to write meaningful code for such a machine: one class of such operations are the so-called "control primitives", i.e., those instructions that are used to code decision-making sequences (e.g. *if, while, repeat,* etc).

[0065] **Pseudocode**

[0066] In the following, pseudocode that uses a notation inspired from the C programming language is utilized to specify informally the semantics of most instructions of the CE **102**. It will be readily appreciated that this kind of description is not intended to limit the expression of the actual implementation of instructions, and that other expressions are also contemplated by the present invention.

[0067] A special pseudo-statement, *forall,* describes actions executed by sets of cells in parallel; its syntax may be expressed as:

- <forall statement> : := forall [(<forall condition>)] <statement>;
- <forall condition> : := marked
  | in searchSpace;

[0068] Consider the following three variants of *forall*:

```
1.      forall
                <action>;
2.      forall (in searchSpace)
                <action>;
3.      forall (marked)
                <action>;
```

[0069] Variant 1 may be utilized to specify an action executed by all the cells in the CM **104**. Variant 2 may be utilized for an action involving all the cells in the search space (see above), while variant 3 applies to cases when all marked cells execute the specified action.

[0070] At the level of each cell, the complete data set needed to execute all the instructions, together with their names used in pseudo-code, are:

- the cell's registers (including associated registers), i.e. *w*, *mark*, r0, r1, ... , rN;.and
- the content of the cell's right and left neighbors, i.e. right_w, right_mark, and left_w, left_mark respectively.

[0071] At the same time, predicates first_mark and last_mark can be evaluated at the cell level; the former is true for the leftmost marked cell in CM, while the latter is true in the case of the rightmost marked cell.

[0072] **Global Management Instructions**

[0073] **ldl** <*value*>: load line immediate; the contents of all CM cells (markers and w registers) are restored from the memory vector selected by the value generated to the input of VAR:

CM = RAM[VAR];

[0074] **stl** <*value*>: store line immediate; the contents of all CM cells (markers and w registers) are saved to the memory vector selected by the value generated to the input of VAR

RAM[VAR] = CM;

[0075] **llim**: left limit; sets the left limit of the search space to the first marked cell. No markers are affected. Note that the left limit is the leftmost cell affected by search/access instructions.

[0076] **rlim**: right limit; sets the right limit of the search space to the first marked cell. No markers are affected. Note that the right limit is the rightmost cell affected by basic search/access instructions.

[0077] **droplim**: remove limits; the left limit is set to the leftmost CM cell, while the right limit is set to the rightmost CM cell. No markers are affected.

[0078] **Search/Access Instructions**

[0079] *Note*: All the instructions described in this section act only within the limits of the search space; arguments are m-1 bit values available in the input register (INR).

[0080] **find** <*value*>: identifies all the cells holding values equal to the argument. For every cell where a match is found, the marker bit of its right neighbour is set to one; all the other marker bits are set to 0:

```
forall (in searchSpace)
        mark = (left_w == INR)? 1: 0;
```

[0081] **match** <*value*>: compares values stored in all marked cells to the argument. If a match is found in a given cell, the marker bit of the following cell is set to 1; all the other marker bits are set to 0:

```
forall (in searchSpace)
        mark = (left_mark && left_w == INR)? 1: 0;
```

[0082] **lfind** <*value*>: find and mark left; identifies all cells holding a value equal to the argument. For every cell where a match is found, the marker bit of its left neighbour is set to one; all the other marker bits are set to 0:

```
forall (in searchSpace)
        mark = (right_w == INR)? 1: 0;
```

[0083] **lmatch** <*value*>: match and mark left; compares values stored in all marked cells to the argument. If a match is found in a given cell, the marker bit of the preceding cell is set to 1; all the other marker bits are set to 0:

```
forall (in searchSpace)
        mark = (right_mark && right_w == INR)? 1: 0;
```

[0084] **markall**: marks all cells in the search space:

```
forall (in searchSpace)
        mark = 1;
```

[0085] **addmark** <*value*>: marks all the cells containing a value equal to the argument; no other markers are affected:

```
forall (in searchSpace) {
        if (w == INR)
        mark = 1;
}
```

[0086] **mark** <*value*>: marks all the cells containing a value equal to the argument; all the other marker bits are set to 0.

```
forall (in searchSpace)
        mark = (w == INR)? 1: 0;
```

[0087] **clr** <*value*>: clears the marker bit of all cells containing a value equal to the argument.

```
forall (in searchSpace) {
        if (w == INR)
                mark = 0;
}
```

[0088] **Marker Manipulation Instructions**

[0089] **clrf**: clear first; clears the first (i.e. leftmost) marker.

```
forall {
        if (first_mark)
        mark = 0;
}
```

[0090] **trace**: duplicates markers leftward.

```
forall {
        if (right_mark)
                mark = 1;
        if (mark)
                mark = 1;
}
```

[0091] **keepl**: keep last; clears all markers except the last (i.e. rightmost) one.

```
forall {
        if (!last_mark)
                mark = 0;
}
```

[0092] **clrl**: clear last; clears the last (i.e. rightmost) marker.

```
forall {
        if (last_mark)
                mark = 0;
}
```

[0093] **left**: shifts all markers one cell to the left.

```
forall
        mark = right_mark;
```

[0094] **right**: shifts all markers one cell to the right.

```
forall
        mark = left_mark;
```

[0095] **cright:** conditional shift right; all markers are shifted one cell to the right unless their right neighbour contains a value equal to the argument, in which case 11...1 is substituted for that value.

```
forall {
        if (left_mark && w == INR) {
                mark = 0;
                w = 11...1;
        }
        if (left_mark && w != INR)
                mark = 1;
        if (mark)
                mark = 0;
        }
```

[0096] **cleft**: conditional shift left; all markers are shifted one cell to the left unless their left neighbour contains a given value, in which case 11...1 is substituted for the value.

```
forall {
        if (right_mark && w == INR) {
                mark = 0;
                w = 11...1;
        }
        if (right_mark && w != INR)
                mark = 1;
                if (mark)
                mark = 0;
        }
```

[0097] **Data store and transfer instructions**

[0098] **nop**: no operation:

[0099] **reset** <*value*>: stores a value in all cells. No markers are affected.

```
forall
        w = INR;
```

[00100] **get**: the value stored in the first marked cell is sent to the CM output and its marker moves one position to the right. No other markers are affected.

```
forall {
        if (first_mark) {
                OUTR = w;
                mark = 0;
        }
        if (left_mark is first_mark)
                mark = 1;
        }
```

[00101] **back**: the value stored in the first marked cell is sent to the CM output and its marker moves one position to the left. No other markers are affected.

```
forall {
    if (first_mark) {
        OUTR = w;
        mark = 0;
    }
    if (right_mark is first_mark)
        mark = 1;
    }
```

[00102] **set** <*value*>: stores a value in the first marked cell. Markers are not affected.

```
forall (marked) {
    if (first_mark)
        w = INR;
}
```

[00103] **setall** <*value*>: stores a value in all marked cells. Markers are not affected.

```
forall (marked)
    w = INR;
```

[00104] **ins** <*value*>: inserts a value before the first marked cell. The contents of all cells to the right of the insertion point are shifted one position to the right. Note that the value initially held in the rightmost cell is lost in the process.

```
forall {
        if (right of first_mark)
                w = left_w;
        if (first_mark)
                w = INR;
        mark = left_mark;
        }
```

[00105] **del**: deletes the value stored in the first marked cell. The cell remains marked and the contents of all cells to the right of the deletion point are shifted one position to the left.

```
forall {
        if (first_mark)
                w = right_w;
        if (right of first_mark) {
                w = right_w;
        mark = right_mark;
        }
}
```

[00106] **cpr**: copy right; for all marked cells, copies the whole cell contents (w register and marker) to the right neighbour.

```
forall {
        if (left_mark)
                w = left_w;
        mark = left_mark;
}
```

[00107] **cpl**: copy left; for all marked cells, copies the whole cell contents (w register and marker) to the left neighbour.

```
forall {
        if (right_mark)
                w = right_w;
        mark = right_mark;
}
```

[00108] **ccpr** <*value*>: conditional copy right; for all marked cells, copies the value held in register w to the right neighbour; markers are also copied, unless a value equal to the argument is stored in w.

```
forall {
        if (left_mark && left_w != INR) {
                w = left_w;
                mark = 1;
}
else
                mark = 0;
}
```

[00109] **ccpl** <*value*>: conditional copy left; for all marked cells, copies the value held in register w to the left neighbour; markers are also copied, unless a value equal to the argument is stored in w.

```
forall {
        if (right_mark && right_w != INR) {
                w = right_w;
                mark = 1;
        }
        else
                mark = 0;
}
```

[00110] **ld** <*address*>: load immediate; for all marked cells, load into w the value held in register <r>, part of the RAM vector selected by the value generated to the input of VAR .

```
forall (marked)
        w = <r>;
```

[00111] **st** < *address*>: store immediate; for all marked cells, move the value held in w to register <r>, part of the RAM vector selected by the value generated to the input of VAR.

```
forall (marked)
        w = <r>;
```

[00112] **Arithmetic & Logic Instructions**

[00113] All arithmetic instructions are carried out on m-2 bit numbers represented in 2's complement: the operand <op> is one of the associated cell registers (part of the RAM vector selected by the value generated to the input of VAR), or a m-2 bit number supplied by the controller:

<op> ::= INR[m-3:0](immediate value) | r(RAM vector element)

[00114] **add** <*op*>: add the operand to the w register of all marked cells. Markers are not affected.

```
forall (marked)
        w += <op>;
```

[00115] **fadd** <*op*>: full add, with the right extension being treated as a carry (see add).

```
forall {
        if (mark)
                w += (<op> + right_w[m-2]);
        if (left_mark)
                w[m-2] = 0;
}
```

[00116] **sub** <*op*>: subtract the operand value from the value stored in the w register of all marked cells. No markers are affected.

```
forall (marked)
        w -= <op>;
```

[00117] **fsub** <*op*>: full subtract, with the right extension being treated as a carry (see sub).

```
forall {
        if (mark)
                w -= (<op> + right_w[m-2]));
        if (left_mark)
                w[m-2] = 0;
}
```

[00118] **half** [<*op*>]: for all marked cells, divide by 2 the register operand and store the result in w. No markers are affected.

```
forall (marked)
        w = {<op>[m-2:m-3], <op>[m-3:1];
```

[00119] **fhalf** [<*op*>]: full half; for all marked cells, divide by 2 the register operand and store the result in w, to which 100...0 is added if the least significant bit of the left cell is 1. Markers are not affected.

```
forall (marked)
        w = {<op>[m-2], left_w[0], <op>[m-3:1];
```

[00120] **lt** *<op>* : less (or equal); for all marked cells, check whether register w holds a value that is less than, or equal to, that held in <op>; if w < op, then the w extension bit is set to 1; the marker bit is set to 0 if op > w.

```
forall (marked) {
        if (w < <op>)
                w[m-2] = 1;
        if (w > <op>)
                mark = 0;
}
```

[00121] **flt** *<op>* : full lt; for all marked cells where w < op or whose left neighbour has the w extension bit set to 1, set the extension bit to 1; the left extension bit is cleared and if w > op the marker is also cleared. This instruction is used in conjunction with test for comparisons on multiple consecutive cells.

```
forall {
        if (mark && (w < <op> | | left_w[m-2]))
                w[m-2] = 1;
        if (right_mark)
                w[m-2] = 0;
        if (mark && w > op && !left_w[m-2])
                mark = 0;
}
```

[00122] **gt** *<op>* : greater (or equal); for all marked cells, check whether register w holds a value that is greater than, or equal to, that held in <op>; if w > op, then the w extension bit is set to 1; the marker bit is set to 0 if w < op.

```
forall (marked) {
        if (w > <op>)
                w[m-2] = 1;
        if (w < <op>)
                mark = 0;
}
```

[00123] **fgt** *<op>* : for all marked cells where w > op or whose left neighbour has the w extension bit set to 1, set the extension bit to 1; the left extension bit is cleared and if w < op the marker is also cleared. This instruction is used in conjunction with test for comparisons on multiple consecutive cells

```
forall {
        if (mark && (w > <op> | | left_w[m-2]))
                w[m-2] = 1;
        if (right_mark)
                w[m-2] = 0;
        if (mark && w < op && !left_w[m-2])
                mark = 0;
}
```

[00124] **test**: For all marked cells containing a value equal to INR the marker bit is set to 0 and if the extension bit of the cell to the left is 1, then register w is assigned 11...1 and the extension bit of the cell to the left is cleared

```
forall {
        if (right_mark && right_w == INR)
                w[m-2] = 0;
        if (mark && w == INR && left_w[m-2])
                w = 11...1;
        else if (mark && w == INR)
                mark = 0;
}
```

[00125] **and** *<op>* : bitwise and; for all marked cells, do a bitwise and between register w and <op>. Markers are not affected.

```
forall (marked)
        w &= <op>;
```

[00126] **or** <op> : bitwise or; for all marked cells, do a bitwise or between register w and <op>. Markers are not affected.

```
forall (marked)
        w |= <op>;
```

[00127] **xor** *<op>* : bitwise xor; for all marked cells, do a bitwise xor between register w and <op>. Markers are not affected.

```
forall (marked)
        w ^= <op>;
```

[00128] **Conditional Instructions**

[00129] The following two instructions use an operand register, <r> (part of the RAM vector selected by the value generated to the input of VAR) and a m-1-bit value from INR.

[00130] <r> ::= w(for register w) | r(RAM vector element)

[00131] **cond** *<value>* [*<r>*] : for all marked cells, check whether there is at least one bit set to 1 after executing a bitwise 'and' operation between the two operands.

```
forall (marked)
        mark = ((<r> & INR) != 0)? 1: 0;
```

[00132] **ncond** *<value>* [*<r>*] : for all marked cells, check whether the result of executing a bitwise 'and' operation between the two operands is 0.

```
forall (marked)
        mark = ((<r> & INR) == 0)? 1: 0;
```

[00133] **Index Instruction**

[00134] **index**: for all marked cells, register w is assigned the value of the cell's relative position with respect to the CM leftmost cell (which has index 0).

[00135] As can be seen from the foregoing descriptions and drawing figures, the present invention provides a novel way to process data in a manner which provides increased processing power with a substantial decrease in processing time, silicon area and power consumption. As discussed, the data processing system of the present invention provides for any given instruction and its operand(s) to be communicated, in parallel, to all CM cells, which execute the instruction within the same clock cycle.

[00136] Yet another inherent advantage of the data processing system of the present invention involves the ability of each cell within the cellular engine to not only simultaneously execute instructions within a single clock cycle, but to also dynamically limit those cells which execute these globally broadcast

instructions via the utilization of both local and global state information. In particular, by utilizing marker bits on an individual cell level, the actual cells within the associative memory are capable of affecting those cells either to the left or right of marked cells in a manner which is heretofore unknown. Therefore, at the system level, the present invention provides for the selective activation, or alteration of the marked state, by associative mechanisms; that is, by the nature or property of the content of the individual cells within the CM **104**, rather than a particular designated location address therein.

[00137] The present invention therefore combines processing and memory at a very intimate level, meaning that an individual cell of the CM **104** never has to access a separate memory block to do its job. Moreover, operands reside in their own local space at the cell level, therefore results are kept in place, saving communication and processing time, silicon area and power.

[00138] It should be noted that some instruction operands are, in fact, broadcast by the CC **100** at the same time as the instruction is globally broadcast.

[00139] While the invention had been described with reference to the preferred embodiments, it will be understood by those skilled in the art that various obvious changes may be made, and equivalents may be substituted for elements thereof, without departing from the essential scope of the present invention. Therefore, it is intended that the invention not be limited to the particular embodiments disclosed, but that the invention includes all embodiments falling within the scope of the appended claims.